Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 262 (2010) 3–16

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

CSL-lean: A Theorem-prover for the Logic of Comparative Concept Similarity

Régis Alenda[1] Nicola Olivetti[2]

LSIS - UMR CNRS 6168
Université Paul Cézanne (Aix-Marseille 3), Domaine Universitaire de Saint-Jérôme
Avenue Escadrille Normandie-Niemen, 13397 Marseille Cedex 20, France

Gian Luca Pozzato[4,3]

Dipartimento di Informatica
Università degli Studi di Torino
c.so Svizzera 185 10149 Torino, Italy

Abstract

The logic $\mathcal{CSL}$ of the comparative concept similarity has been introduced by Sheremet, Tishkovsky, Wolter and Zakharyaschev to capture a form of qualitative similarity comparison between concepts and/or objects. In this logic we can formulate assertions of the form "objects A are more similar to B than to C". This kind of assertions can be added to an ontology to express qualitative comparisons between concepts. In this work we present CSL-lean, the first theorem-prover for this logic. It is a direct Prolog implementation of a tableaux-based decision procedure recently proposed for this logic. The Prolog program is inspired by the lean-methodology. CSL-lean also contains a graphical interface written in Java and it is available for free download at http://www.di.unito.it/~pozzato/csllean/.

Keywords: Comparative concept similarity, Tableaux Calculi, Logic Programming, lean methodology.

1 Introduction

The logics of comparative concept similarity $\mathcal{CSL}$ have been introduced in [15] to capture a form of qualitative comparison between concept instances. In these logics we can express assertions of the form: "Renault Clio is more similar to Peugeot 207 than to VW Golf", "Marseilles is more similar to Barcelona than to Naples". These logics may find an application in ontology languages, whose logical base is

[1] Email: regis.alenda@lsis.org

[2] Email: nicola.olivetti@univ-cezanne.fr

[3] Email: pozzato@di.unito.it

[4] The author has been partially supported by Regione Piemonte, Project "ICT4Law - *ICT Converging on Law: Next Generation Services for Citizens, Enterprises, Public Administration and Policymakers*".

doi:10.1016/j.entcs.2010.04.002

provided by Description Logics (DL), allowing concept definitions based on proximity/similarity measures. For instance ([15]), the color "Reddish" may be defined as a color which is more similar to a prototypical "Red" than to any other color (in some color model as RGB). The aim is to develop a language in which logical classification provided by standard DL is integrated with classification mechanisms based on calculation of proximity measures. The latter is typical for instance of domains like bio-informatics or linguistics.

The logic $\mathcal{CSL}$ contains formulas (or concepts) defined by boolean operators and a single binary modal connective $\trianglelefteq$ expressing comparative similarity [5]. In this language the above examples can be encoded (using nominals as needed) as follows:

(1) $reddish \equiv \{red\} \trianglelefteq \{green, \ldots, black\}$,
(2) $Clio \sqsubseteq (Peugeot207 \trianglelefteq Golf)$,
(3) $(\{barcelona\} \trianglelefteq \{naples\})(marseilles)$,

The semantics of $\mathcal{CSL}$ is defined in terms of distance spaces, that is to say structures equipped with a distance function d, whose properties may vary according to the logic under consideration. In this setting, the evaluation of $A \trianglelefteq B$ can be informally stated as follows: x satisfies $A \trianglelefteq B$ iff $d(x, A) < d(x, B)$ meaning that the object x is an instance of the concept $A \trianglelefteq B$ (i.e. it belongs to things that are more similar to A than to B) if x is strictly closer to A-objects than to B-objects according to distance function d, where the distance of an object to a set of objects is defined as the *infimum* of the distances to each object in the set. In [15,17,8,16], the authors have investigated the logic $\mathcal{CSL}$ with respect to different classes of distance models, see [17] for a survey of results about decidability, complexity, expressivity, and axiomatisation. Remarkably it is shown that $\mathcal{CSL}$ is undecidable over subspaces of the reals.

A particular dialect of $\mathcal{CSL}$ has been investigated in great detail [15,1]: namely the case in which the semantics is restricted to *minspaces*, the latter being spaces where the infimum of a set of distances is actually their *minimum*. The *minspace* property entails the restriction to spaces where the distance function is *discrete*. This requirement does not seem incompatible with the purpose of representing *qualitative* similarity comparisons. We consider here $\mathcal{CSL}$ under the minspace semantics.

As shown in [1], the semantics based on minspace can be formulated equivalently in terms of preferential structures, that is to say Kripke models equipped with a family of strict partial (pre)-orders $y \prec_x z$ indexed on objects x [10,18], whose intended meaning is that x is more similar to y than to z.

A decision procedure for this logic in the form of tableaux calculus is presented in [1]. Termination is obtained by adopting a standard strategy (controlling the application order of the rules) and by suitable blocking conditions. In [2], the tableaux procedure is extended to a simple description logic language containing also ABOX and nominals.

[5] In a more general setting, the language might contain several $\trianglelefteq_{\text{Feature}}$ where each Feature corresponds to a specific distance function d_{Feature} measuring the similarity of objects with respect to one Feature (size, price, power, taste, color...).

In this paper we present CSL-lean, the first theorem prover for $\mathcal{CSL}$ logic. It is a (relatively small) Prolog program inspired by the "lean" methodology: in a few words, the program implements just a "prove" predicate, each clause of its definition corresponds to a tableau rule and the search mechanism is given for free by the Prolog search engine. The preliminary version we present here does not make use of any sophisticated data structure or optimization technique. It only uses basic Prolog predicates and implements directly the blocking conditions to obtain termination. Performances are nonetheless encouraging. Some improvements and further extensions are discussed in the conclusions.

2 The Logic $\mathcal{CSL}$

The language $\mathcal{L}_{\mathcal{CSL}}$ of $\mathcal{CSL}$ is generated from a (countable) set of propositional variables $V_1, V_2, \ldots \in \mathcal{V}_p$ by ordinary propositional connectives plus $\leftleftarrows$:
$A, B \ ::= \ \bot \mid V_i \mid \neg A \mid A \sqcap B \mid A \leftleftarrows B$ (where $V_i \in \mathcal{V}_p$).

The semantics of $\mathcal{CSL}$ is defined in terms of models based on *distance spaces*. A distance space is a pair (Δ, d) where Δ is a non-empty set, and $d : \Delta \times \Delta \to \mathbb{R}^{\geq 0}$ is a *distance function* satisfying the following condition [6]

$$\forall x, y \in \Delta, \quad d(x, y) = 0 \text{ iff } x = y \ . \tag{ID}$$

The distance between an object w and a non-empty subset X of Δ is defined by $d(w, X) = \inf\{d(w, x) \mid x \in X\}$. If $X = \emptyset$, then $d(w, X) = \infty$. If for every object w and for every (non-empty) subset X we have the following property

$$\inf\{d(w, x) \mid x \in X\} = \min\{d(w, x) \mid x \in X\} \ , \tag{MIN}$$

we say that (Δ, d) is a *minspace*.

We next define $\mathcal{CSL}$-distance models as Kripke models based on distance spaces:

Definition 2.1 [$\mathcal{CSL}$-distance model]
A $\mathcal{CSL}$-distance model is a triple $\mathcal{M} = (\Delta, d, .^{\mathcal{M}})$ where:

- Δ is a non-empty set of *objects*.
- d is a distance on Δ (so that (Δ, d) is a distance space).
- $.^{\mathcal{M}} : \mathcal{V}_p \to 2^{\Delta}$ is the *evaluation function* which assigns to each propositional variable V_i a set $V_i^{\mathcal{M}} \subseteq \Delta$. We further stipulate:
  $\bot^{\mathcal{M}} = \emptyset \ , \qquad (\neg C)^{\mathcal{M}} = \Delta - C^{\mathcal{M}} \ , \qquad (C \sqcap D)^{\mathcal{M}} = C^{\mathcal{M}} \cap D^{\mathcal{M}} \ ,$
  $(C \leftleftarrows D)^{\mathcal{M}} = \{w \in \Delta \mid d(w, C^{\mathcal{M}}) < d(w, D^{\mathcal{M}})\} \ .$

If (Δ, d) is a minspace, $\mathcal{M}$ is called a *$\mathcal{CSL}$-minspace model*. We say that a formula A is *valid in a model* $\mathcal{M}$ if $A^{\mathcal{M}} = \Delta$. We say that a formula A is *valid* if A is valid in every $\mathcal{CSL}$-distance model ($\mathcal{CSL}$-minspace model).

[6] Two well-known properties of symmetry and triangle inequality may be additionally considered; for a discussion we refer to [1].

From now on we restrict our consideration to minspace models, thus we write model instead of minspace model.

$\mathcal{CSL}$ is a logic of pure qualitative comparisons, this motivates an alternative semantics where the distance function is replaced by a family of comparison relations, one for each object. This semantics is called *preferential* semantics, similarly to the semantics of conditional logics [11,10]. Preferential structures are equipped with a family of strict pre-orders indexed on objects: for three objects, $x \prec_w y$ may be interpreted as w is more similar to x than to y. We assume the following conditions on $x \prec_w y$:

(i) *(modularity)* $\forall x, y, z \in \Delta$, $(x \prec_w y) \rightarrow (z \prec_w y \ \vee \ x \prec_w z)$.

(ii) *(centering)* $\forall x \in \Delta$, $x = w \ \vee \ w \prec_w x$.

(iii) *(Limit Assumption)* $\forall X \subseteq \Delta$, $X \neq \emptyset \ \rightarrow \ \min_{\prec_w}(X) \neq \emptyset$, where $\min_{\prec_w}(X) = \{y \in X \mid \forall z(z \prec_w y \rightarrow z \notin X)\}$.

Definition 2.2 [$\mathcal{CSL}$-preferential model] A $\mathcal{CSL}$-preferential model is a triple $\mathcal{M} = (\Delta, (\prec_w)_{w\in\Delta}, .^{\mathcal{M}})$ where:

- Δ is a non-empty set of *objects* (or *possible worlds*).
- $(\prec_w)_{w\in\Delta}$ is a family of *preferential relation* satisfying the above conditions (i), (ii), and (iii).
- $.^{\mathcal{M}}$ is the evaluation function defined as in definition 2.1, except for $\leftleftarrows$:

$$(A \leftleftarrows B)^{\mathcal{M}} = \left\{w \in \Delta \middle| \exists x \in A^{\mathcal{M}} \text{ such that } \forall y \in B^{\mathcal{M}},\ x \prec_w y\right\} \ .$$

Validity is defined in the same way as in Definition 2.1.

Distance models and preferential models provide an equivalent semantics of $\mathcal{CSL}$ (see [1]) in the sense that for each $\mathcal{CSL}$-distance minspace model there is an equivalent $\mathcal{CSL}$-preferential model and vice versa.

A sound and complete axiomatization of $\mathcal{CSL}$ is provided in [1] and it comprises the following axioms and rules:

(1) $\neg(A \leftleftarrows B) \sqcup \neg(B \leftleftarrows A)$

(2) $(A \leftleftarrows B) \rightarrow (A \leftleftarrows C) \sqcup (C \leftleftarrows B)$

(3) $A \sqcap \neg B \rightarrow (A \leftleftarrows B)$

(4) $(A \leftleftarrows B) \rightarrow \neg B$

(5) $(A \leftleftarrows B) \sqcap (A \leftleftarrows C) \rightarrow (A \leftleftarrows (B \sqcup C))$

(6) $(A \leftleftarrows \bot) \rightarrow \neg(\neg(A \leftleftarrows \bot) \leftleftarrows \bot)$

(Mon) $\dfrac{\vdash (A \rightarrow B)}{\vdash (A \leftleftarrows C) \rightarrow (B \leftleftarrows C)}$

$(Taut)$ Classical tautologies and rules.

3 A Tableaux Calculus for $\mathcal{CSL}$

In [1] it is presented a tableau calculus for $\mathcal{CSL}$ that we recall here. A tableau is identified with a set of sets of formulas $\Gamma_1, \ldots, \Gamma_n$. Each Γ_i is called a *tableau set*. The calculus is called $\mathbf{T}\mathcal{CSL}$ and it makes use of labels to represent objects of the domain. To understand the treatment of $\leftrightarrows$, let us consider formulas $(A \leftrightarrows B)$ and $\neg(A \leftrightarrows B)$ under preferential semantics. We have:

$$w \in (A \leftrightarrows B)^{\mathcal{M}} \text{ iff } \exists x(x \in A^{\mathcal{M}} \wedge \forall z(z \in B^{\mathcal{M}} \rightarrow x \prec_w z)) \ .$$

But in minspace models, the right part is equivalent to:

$$w \in (A \leftrightarrows B)^{\mathcal{M}} \text{ iff } \exists u \in A^{\mathcal{M}} \text{ and } \forall y(y \in B^{\mathcal{M}} \rightarrow \exists x(x \in A^{\mathcal{M}} \wedge x \prec_w y)) \ .$$

We can hence introduce a pseudo-modality $\Box_w$ indexed on objects:

$$x \in (\Box_w A)^{\mathcal{M}} \text{ iff } \forall y(y \prec_w x \rightarrow y \in A^{\mathcal{M}}) \ .$$

Thus we get the equivalence:

$$w \in (A \leftrightarrows B)^{\mathcal{M}} \text{ iff } A^{\mathcal{M}} \neq \emptyset \text{ and } \forall y(y \notin B^{\mathcal{M}} \text{ or } y \in (\neg\Box_w\neg A)^{\mathcal{M}}) \ .$$

This equivalence allows us to decompose $\leftrightarrows$-formulas in an analytic way. The tableau rules make also use of a universal modality $\Box$ (and its negation). The language of tableaux comprises the following kind of formulas: $x : A, x : (\neg)\Box\neg A, x : (\neg)\Box_y\neg A, x <_y z$, where x, y, z are labels and A is a $\mathcal{CSL}$-formula. The meaning of $x : A$ is the obvious one: $x \in A^{\mathcal{M}}$. The reading of the rules is the following: we apply a rule

$$\frac{\Gamma[E_1, \ldots, E_k]}{\Gamma_1 \mid \ \ldots \ \mid \Gamma_n}$$

to a tableau set Γ if each formula E_i $(1 \leq i \leq k)$ is in Γ. We then replace Γ with any tableau set $\Gamma_1, \ldots, \Gamma_n$. As usual, we let Γ, A stand for $\Gamma \cup \{A\}$, where A is a tableau formula. The tableaux rules are shown in Figure 1, we refer to [1] [7] for a detailed explanation of the rules.

In [1] it is shown that the calculus $\mathbf{T}\mathcal{CSL}$ is sound and complete with respect to the preferential semantics, whence with respect to minspace models.

Theorem 3.1 *A formula $A \in \mathcal{L}_{\mathcal{CSL}}$ is satisfiable with respect to preferential semantics iff it is contained in an open saturated tableau set.*

The calculus $\mathbf{T}\mathcal{CSL}$ presented above can lead to non-terminating computations due to the interplay between the dynamic rules (namely $(F2 \leftrightarrows)$, $(F\Box)$ and $(F2\Box_x)$) and the static rules. It can be made terminating by defining a systematic procedure for applying the rules and by introducing appropriate *blocking conditions*. The

[7] In the formulation of the rules for $(F1 \leftrightarrows)$ and $(F2 \leftrightarrows)$ given in [1] the formula $x : \neg\Box\neg A$ in the consequent of $(F1 \leftrightarrows)$ and in the premise of $(F2 \leftrightarrows)$ was erroneously omitted. The formulation given here is the correct one.

$$(T\sqcap)\quad \frac{\Gamma[x : A \sqcap B]}{\Gamma,\ x : A,\ x : B} \qquad (F\sqcap)\quad \frac{\Gamma[x : \neg(A \sqcap B)]}{\Gamma,\ x : \neg A \mid \Gamma,\ x : \neg B}$$

$$(T \leftleftarrows)(*)\quad \frac{\Gamma[x : A \leftleftarrows B]}{\Gamma, x : \neg\Box\neg A,\ y : \neg B \mid \Gamma, y : B, y : \neg\Box_x\neg A} \qquad (NEG)\quad \frac{\Gamma[x : \neg\neg A]}{\Gamma, x : A}$$

$$(F1 \leftleftarrows)\quad \frac{\Gamma[x : \neg(A \leftleftarrows B)]}{\Gamma, x : \Box\neg A \mid \Gamma, x : B \mid \Gamma, x : \neg A,\ x : \neg B,\ x : \neg\Box\neg A}$$

$$(F2 \leftleftarrows)(**)\quad \frac{\Gamma[x : \neg(A \leftleftarrows B),\ x : \neg A,\ x : \neg B,\ x : \neg\Box\neg A]}{\Gamma,\ y : B,\ y : \Box_x\neg A} \qquad (F1\Box_x)\quad \frac{\Gamma[z : \neg\Box_x\neg A]}{\Gamma, x : \neg A \mid \Gamma, x : A}$$

$$(T\Box_x)(*)\quad \frac{\Gamma[z : \Box_x\neg A, y <_x z]}{\Gamma,\ y : \neg A,\ y : \Box_x\neg A} \qquad (F2\Box_x)(**)\quad \frac{\Gamma[z : \neg\Box_x\neg A,\ x : \neg A]}{\Gamma,\ y <_x z,\ y : A,\ y : \Box_x\neg A}$$

$$(T\Box)(*)\quad \frac{\Gamma[x : \Box\neg A]}{\Gamma,\ y : \neg A, y : \Box\neg A} \qquad (F\Box)(**)\quad \frac{\Gamma[x : \neg\Box\neg A]}{\Gamma,\ y : A}$$

$$(Mod)(*)\quad \frac{\Gamma[z <_x u]}{\Gamma, z <_x y \mid \Gamma, y <_x u} \qquad (Cent)(***)\ \frac{\Gamma}{\Gamma, x <_x y \mid \Gamma[x/y]}$$

(*) y is a label occurring in Γ. (**) y is a new label not occurring in Γ. (***) x and y are two distinct labels occurring in Γ.

Fig. 1. The tableau calculus $\mathbf{T}\mathcal{CSL}$.

systematic procedure simply prescribes to apply the static rules as far as possible before the applying dynamic rules. To prevent the generation of an infinite tableau set however some restrictions on the application of the rules are needed. The non-trivial restrictions are those ones on $(F2 \leftleftarrows)$ and $(F2\Box_x)$, they are called as usual *blocking conditions* and prevent the generation of infinitely many labels by performing a kind of loop-checking.

In order to define the blocking restrictions, we first define a total ordering $\sqsubset$ on the labels of a tableau set such that $x \sqsubset y$ for all labels x that are already in the tableau set when y is introduced. If $x \sqsubset y$, we will say that x is older than y. We further define $\mathrm{Box}^+_{\Gamma,x,y} = \{\Box_x\neg A \mid y : \Box_x\neg A \in \Gamma\}$ and $\Pi_\Gamma(x) = \{A \mid A \in \mathcal{L}_{\mathcal{CSL}}$ and $x : A \in \Gamma\}$. The restrictions on rules application and the strategy are summarised in the next definition.

Definition 3.2 • *(Static and dynamic rules)* We call *dynamic* the following rules: $(F2 \leftleftarrows)$, $(F2\Box_x)$ and $(F\Box)$. We call *static* all the other rules.

• *(Rules restrictions)*

(i) Do not apply a static rule to Γ if at least one of the consequences is already in it.

(ii) Do not apply the rule $(F2 \leftleftarrows)$ to a $x : \neg(A \leftleftarrows B), x : \neg A, x : \neg B$
  (a) if there exists some label y in Γ such that $y : B$ and $y : \Box_x \neg A$ are in Γ.
  (b) if there exists some label u such that $u \sqsubset x$ and $\Pi_\Gamma(x) \subseteq \Pi_\Gamma(u)$.
(iii) Do not apply the rule $(F2\Box_x)$ to a $z : \neg\Box_x\neg A, x : \neg A$
  (a) if there exists some label y in Γ such that $y <_x z$, $y : A$ and $y : \Box_x \neg A$ are in Γ.
  (b) if there exists some label u in Γ such that $u \sqsubset x$ and $\Pi_\Gamma(x) \subseteq \Pi_\Gamma(u)$.
  (c) if there exists some label v in Γ such that $v \sqsubset z$ and $v : \neg\Box_x\neg A \in \Gamma$ and $\mathrm{Box}^+_{\Gamma,x,z} \subseteq \mathrm{Box}^+_{\Gamma,x,v}$.
(iv) Do not apply the rule $(F\Box)$ to a $x : \neg\Box\neg A$ in Γ if there exists some label y such that $y : A$ is in Γ.

- *(Systematic procedure)* (1) Apply static rules as far as possible. (2) Apply a (non blocked) dynamic rule to some formula labelled x only if no dynamic rule is applicable to a formula labelled y, such that $y \sqsubset x$.

It can be shown (see [1], Theorems 13 and 14) that (i) a tableau initialized with a $\mathcal{CSL}$-formula and expanded according to Definition 3.2 always terminates and (ii) the restrictions preserve the completeness of the calculus. Thus the calculus provides a decision procedure for the logic, running in NEXPTIME. It follows that our tableaux calculus is not worst case optimal because $\mathcal{CSL}$ over minspaces is ExpTime-complete [15].

4 The theorem-prover CSL-lean

In this section we present CSL-lean, an implementation of the tableau calculus $\mathbf{T}\mathcal{CSL}$ of Figure 1 above. It is a Prolog program inspired by the "lean" methodology originally introduced by leanT^AP ([3,4]). The program CSL-lean implements a predicate `csl_lean_aux`. Each clause of this predicate represents a tableau rule or axiom. Proof search is provided for free by the mere depth-first search mechanism of Prolog, without any meta-level algorithm of search strategy. In this way, the philosophy underlaying the "lean" methodology is "to achieve maximal efficiency from minimal means" [3], that is to say to write short programs and exploit the power of Prolog's engine as much as possible. It is worth noticing that CSL-lean is only inspired to the "lean" methodology, but it does not fit its style in a rigorous manner.

4.1 The main predicate

The tableau calculus $\mathbf{T}\mathcal{CSL}$ is implemented by the predicate:

```
csl_lean_aux(Gamma, Labels, UsedBox, UsedIBOX, UsedCCS, UsedNIBOX,
             UsedNCCS, UsedNBOX, PrefRel, ProofTree).
```

which succeeds if and only if Γ is unsatisifable, where the set of formulas Γ is partitioned into the Prolog lists `Gamma` and `PrefRel`. `Gamma` is a list containing, for each label x occurring in a tableau set, a pair `[X,Formulas]` where `Formulas` is the list of *all* formulas F such that $x : F \in \Gamma$. In other words, formulas of each node are grouped by the labels. The list `PrefRel` contains lists `[x,y,z]`, representing

formulas of the form $x <_y z$. For instance, the set of formulas $x : \Box\neg A, y : B, x : C, x <_x y, y : \Box_x\neg B$ is represented by the list `Gamma`

```
[[x,[box (neg a),c]],[y,[b,ibox(x,neg a)]]]
```

plus the following list `PrefRel`

```
[[x,x,y]]
```

Labels are represented by Prolog's constants. The argument `Labels` is the list of labels introduced in the current tableau set. It is to be noted that this list is sorted (by construction) in anti-chronological order. The predicate `NewLabel` is called by the dynamic rules to create a new label, which will be added on the head of the list. So the list `Labels` is also used to represent the chronological order $\sqsubset$ on the labels: we have $x \sqsubset y$ (that is to say x is *older* than y) if x appears after y in `Labels`.

As an example, to prove whether $\neg(\Box P) \vee (P \leftleftarrows P)$ is unsatisfiable or not, one queries CSL-lean with the goal

```
csl_lean_aux([[x,neg (box p) or (p < <- p)]], [x], [], [],
             [], [], [], [], [], ProofTree).
```

`ProofTree` is the only output parameter. When the initial set of formulas Γ is unsatisfiable, then `ProofTree` matches with a Prolog functor `tree` representing a closed tableau for Γ.

4.2 Preventing redundant applications of the rules

In order to ensure the termination of the calculus, the prover CSL-lean implements the rule restrictions and the strategy of Definition 3.2. The application of the rules dealing with classical connectives is controlled in a standard way, namely by allowing their application only if the current set of formulas does not contain the formula(s) introduced by the rules in their conclusion(s). For instance, (NEG) is applicable to $\Gamma[x : \neg\neg A]$ only if `[x, A]` does not belong to `Gamma`. For the rules $(F2 \leftleftarrows)$, $(F2\Box_x)$ and $(F\Box)$, Prolog lists `UsedNIBOX`, `UsedNCCS`, and `UsedNBOX` are used. They contain the list of labelled formulas to which the corresponding rule has already been applied in the current tableau set. For instance, if $(F2\Box_x)$ has already been applied to $\Gamma[z : \neg\Box_x\neg A, x : \neg A]$, then the list `UsedNIBOX` contains `[z,neg ibox (x, neg a)]`, and the clause implementing the rule is no longer applicable to that formula in that tableau set. `UsedBox`, `UsedIBox`, and `UsedCCS` are lists of pairs of the kind `[Formula,ListOfLabels]`, where `ListOfLabels` keeps track of the labels already used to apply, in the current tableau set, the rule associated to the top-level connective in `Formula`. These lists are also used in order to ensure the termination of the proof search, by restricting the choice of the label to those not belonging to `ListOfLabels`. For instance, `UsedBox` is a list of pairs of the form `[box (neg A), ListOfLabels]`. The rule $(T\Box)$ is then applied to a formula $x : \Box\neg A$ by introducing $y : \neg A$ and $y : \Box\neg A$ in the conclusion by choosing a label y only if `y` does not belong to `ListOfLabels`.

4.3 Handling the universal *rules*

The "universal[8]" rules $(T\Box)$, $(T\Box_x)$, and $(T \leftleftarrows)$ are split in two clauses: the first one represents the first application of the rule to a formula $x : \Box\neg A$, $y : \Box_x\neg A$ or $x : A \leftleftarrows B$ (respectively), i.e when these formulas are not in `UsedBox`, `UsedIBOX` or `UsedCCS` (respectively). The rule is applied once to the formula, and then added to the corresponding "used formulas" list. The second clause selects a used formula from one of these lists, looks for a label to which the corresponding rule has not been applied, applies the rule to it, and mark this label as used for the corresponding formula. It is to be noted that the first clause implementing the application of the rule $(T \leftleftarrows)$ to a formula $x : A \leftleftarrows B$ applies it first to the original label x, and thus does not branch: only the branch with the tableau set $\Gamma, x : \neg B, x : \neg\Box\neg A$ is generated, as the other would be trivially closed due to the presence of the formula $x : \neg\Box_x\neg A$ in the new tableau set.

4.4 Handling the dynamic rules

The clauses for the dynamic rules implements the restrictions (ii)(b), (iii)(b), (iii)(c) and (iv) of definition 3.2. The restriction (iv) is easy: before applying the rule $(F\Box)$ to a formula $x : \neg\Box\neg A$, we check whether there is no label z such that $z : A$ is in the current tableau set. This is done by the predicate `exist_formula(A,Gamma)`: if this predicate succeeds, the rule is not applied.

For the other restrictions, two predicates are used: `is_blocked_2b3b(X, Gamma, Labels)` which succeeds if there is a label u older than x such that $\Pi_\Gamma(x) \subseteq \Pi_\Gamma(u)$, and `is_blocked_3c(Z, neg ibox(X, neg A), Gamma, Labels)` which succeeds if there is a label u older than z such that $u : \neg\Box_x\neg A \in \Gamma$ and $\mathrm{Box}^+_{x,z} \subseteq \mathrm{Box}^+_{x,u}$. As mentioned before, the ordering of labels is directly encoded by the list `Labels`. $\Pi_\Gamma(x)$ is computed by the predicate `extract_CSL_formulas(XFormulas, PiX)`, where `XFormulas`) is the list of formulas labelled by x, and $\mathrm{Box}^+_{x,u}$ is computed by `extract_box_X(X, UFormulas, UBox)`.

Once a non-blocked dynamic rule has been applied to a formula $x : \neg\Box\neg A$, $x : \neg(A \leftleftarrows B)$ or $z : \neg\Box_x\neg A$, this formula is added to the list `UsedNBOX`, `UsedNCCS` or `UsedNIBOX` (respectively), to prevent multiple applications of these rules to the same formula. Therefore, an application of a dynamic rule may lead to create at most one new label.

To give an example, the clause implementing the rule $(F\Box_x 2)$ is:

```
csl_lean_aux(Gamma, Labels, UsedBox, UsedIBOX, UsedCCS, UsedNIBOX,
          UsedNCCS, UsedNBOX, PrefRel,
          tree(fibox2,[Z,neg ibox (X,neg A)],SubTree)):-
     member([Z,Formulas],Gamma),
     member(neg ibox(X, neg A),Formulas),
     \+memberchk([Z,neg ibox(X, neg A)], UsedNIBOX),
     memberchk([X, XFormulas],Gamma),
```

[8] We call them *universal* because they contain a kind of implicit universal quantification.

```
memberchk(neg A, XFormulas),
\+is_blocked_2b3b(X, Gamma, Labels),
\+is_blocked_3c(Z, neg ibox(X, neg A), Gamma, Labels),!,
newLabel(Labels,Y),
csl_lean_aux([[Y, [A,ibox(X, neg A)]]|Gamma],[Y|Labels],UsedBox,
     UsedIBOX, UsedCCS, [[Z, neg ibox(X, neg A)]|UsedNIBOX],
     UsedNCCS, UsedNBOX, [[Y,X,Z]|PrefRel], SubTree).
```

The first two predicates seek for a formula $z : \neg\Box_x\neg A$. If one is found, then the third predicate checks whether the rule has already been applied to it or not. If not, we test the other precondition of the rule: is there a formula $x : \neg A$ in Γ (predicates 4 and 5)? Then we check the blocking conditions: the sixth predicate tests the restriction (iii)(b), and the seventh the restriction (iii)(c). If the formula is not blocked, then we proceed with the application of the rule.

4.5 Search strategy and further improvements

To search a closed tableau for a set of formulas Γ, CSL-lean proceeds as follows. First of all, if Γ is an axiom, the goal will succeed immediately by using the clauses for the axioms. If it is not, then the first applicable rule will be chosen, e.g. if `Gamma` contains a formula `[X,neg neg A]`, then the clause for (NEG) rule will be used, invoking `csl_lean_aux` on its unique conclusion. CSL-lean proceeds in a similar way for the other rules. The ordering of the rules is such that it implements the systematic procedure (static rules first, dynamic ones last, as seen in Definition 3.2.), together with some efficiency improvements. Experiments have shown that some small change in the rules application's order can have a significant impact on the performances. Static rules can be divided in two groups: non-branching ones, and branching ones. Branching rules may strongly affect the performances. Thus we first saturate Γ with static non-branching rules, then non-branching ones, and finally we apply dynamic rules. Thus the ordering of clauses that seems to work better and that we have adopted in the current version is the following: (NEG), $(T\sqcap)$, $(T\Box)$, $(T \leftleftarrows)$, $(T\Box_x)$, $(F \leftleftarrows 1)$, $(F\Box_x 1)$, $(T\sqcup)$, (Cent), (Mod), $(F \leftleftarrows 2)$, $(F\Box_x 2)$ and finally $(F\Box)$.

We further notice that the rule $(F\Box)$ is the last one to be applied since the applications of the two other dynamic rules can create the label and the formula that fulfills $(F\Box)$, making its application redundant (blocking condition (iv)). We need to control tightly the creation of new labels, in particular, because they may provoke a combinatory explosion due to the rules (Cent) and (Mod).

The rules of (Cent) and (Mod) have also a heavy impact on performances. Concerning these rules, a small optimisation is used: a preferential relation $\prec_x$ is generated only if a formula $y : \Box_x\neg A$ is in Γ (as the preferential relation is only needed for propagating $\Box_x$ formulas).

5 Statistics and Performances

The performance of CSL-lean are promising. We have tested it running SICStus Prolog 4.0.2 on an Apple MacBook Pro 3.06 GHz Intel Core 2 Duo machine (4GB RAM), obtaining the results presented in the following tables.

First, we have tested CSL-lean by randomly generating generic formulas (valid, unsatisfiable and satisfiable). The test samples have been generated by fixing two parameters:

- the number of propositional variables involved in the generated formulas;
- the *depth* of connectives, i.e. the maximum level of nesting of connectives in the generated formulas.

The table below shows the number of proofs successfully completed (with either a positive or a negative answer) with respect to a fixed time limit. The first column shows the two parameters taken into account in generating test formulas mentioned above, namely the number of propositional variables and the depth of the formulas. For each row, we have considered 1000 test samples.

	Time to succeed				
Prop. vars - Depth	$1ms$	$10ms$	$1s$	$2s$	$10s$
2 - 2	975	978	992	1000	1000
2 - 4	855	862	873	910	920
3 - 4	873	881	886	887	887
5 - 7	805	832	840	845	846

We have also tested CSL-lean over some unsatisfiable formulas, most of them obtained from negated instances of some axioms of $\mathcal{CSL}$ [9]. The following table shows the number of successes (with respect to 34 unsatisfiable formulas) within a fixed time limit:

$1ms$	$10ms$	$1s$	$1.5s$
18 $(52,94\%)$	18 $(52,94\%)$	20 $(58,82\%)$	31 $(91,18\%)$

As an example, CSL-lean answer in less than 1 millisecond on the following unsatisfiable formula:

$$(P \leftleftarrows (Q \leftleftarrows Q)) \leftleftarrows (P \leftleftarrows (Q \leftleftarrows Q))$$

[9] The set of unsatisfiable formulas of $\mathcal{CSL}$ used to test CSL-lean is available at http://www.di.unito.it/~pozzato/csllean/benchmark.txt.

Finally, although the logics are different, we have compared the performances of CSL-lean with two other theorem provers recently introduced, namely CondLean [14,12,13], a theorem prover for Conditional Logics, and KLMLean [5], a theorem prover for Kraus, Lehmann, and Magidor (KLM) Preferential logics [7,9]. The table shows the number of answers given by each theorem prover on a set of 90 formulas within 2 seconds. Those formulas are group by their depth:

		Theorem prover	
Depth	CSL-lean	CondLean (CK)	KLMLean (P)
2	82	79	79
4	76	72	69
7	67	62	58

The experimental results listed above show that the performances of CSL-lean are as good as those ones of other similar "lean" provers (for the respective logics) and can be considered promising. However, we intend to improve the performances of CSL-lean by experimenting standard optimization techniques.

6 Conclusions

In this paper we have described CSL-lean. To the best of our knowledge it is the first theorem prover for $\mathcal{CSL}$ Logics. To this concern, in [6] a tableau algorithm and a theorem prover is proposed for a logic of metric spaces comprising several kinds of distance quantifiers, although this logic is related to $\mathcal{CSL}$, it cannot encode the $\leftleftarrows$ operator. Our prover is based on the tableau calculus presented in [1] and implemented following the simple "lean" methodology of leanT^AP ([3,4]). CSL-lean is only inspired to the "lean" methodology, but it does not fit this style in a rigorous manner. The main differences between our implementation and a "*really-lean*" one are the following:

- CSL-lean makes use of some auxiliary predicates, such as `is_blocked`, `newLabel`, and `member`, whereas leanT^AP only relies on Prolog's clause indexing scheme and backtracking;
- the first argument of the predicate `prove` in leanT^AP is the next formula to be processed, which is always the leftmost formula in a single-sided sequent; this allows it to use the first-argument indexing refinements available in SICStus Prolog. CSL-lean does not present this characteristic, so it cannot take advantage of this refinement.

CSL-lean is also inspired by CondLean [14,12,13], a theorem prover for Conditional Logics, and KLMLean [5], a theorem prover for Kraus, Lehmann, and Magidor (KLM) nonmonotonic logics [7,9]. CSL-lean also contains a graphical interface writ-

ten in Java and it is available for free download at

http://www.di.unito.it/~pozzato/csllean/

Even if there is not a set of benchmarks of reference, the tests show that the performances of CSL-lean are encouraging.

In further research we intend to explore several directions. First of all the prover could be made more efficient. Of course many ad-hoc optimisations are possible. But as we have recalled the tableau calculus $\mathbf{T}\mathcal{CSL}$ itself on which CSL-lean is based is not optimal. A theoretical study is then required about how to obtain an optimal prover. This could be the base of a substantially more efficient version. In order to improve efficiency some form of caching is needed to avoid the repeated generation of the same node and to early detect open and closed saturated nodes.

Another direction of research is the extension of the prover to a richer language of description logic family, first of all with nominals and then with quantified role restrictions. Preliminary results in [2] show that it is possible to extend the tableau method with nominals (ABOX and enumerative concepts).

Finally we can think of developing a similar prover for other $\mathcal{CSL}$ logics, notably for the one corresponding to symmetric minspaces. To this aim we should first find a suitable tableau calculus for it, what we are currently investigating.

References

[1] Alenda, R., N. Olivetti and C. Schwind, *Comparative concept similarity over minspaces: Axiomatisation and tableaux calculus*, in: M. Giese and A. Waaler, editors, *Proceedings of TABLEAUX 2009 (18th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods)*, LNAI **5607** (2009), pp. 17–31.

[2] Alenda, R., N. Olivetti and C. Schwind, *Reasoning about concept similarity in ontologies: a first step*, Journées Nationales de l'IA Fondamentale (2009), available at `www.lsis.org/olivetti/TR09/JIAF09-Alenda-Olivetti-Schwind.pdf`.

[3] Beckert, B. and J. Posegga, *leantap: Lean tableau-based deduction*, Journal of Automated Reasoning **15** (1995), pp. 339–358.

[4] Fitting, M., *leantap revisited*, Journal of Logic and Computation **8** (1998), pp. 33–47.

[5] Giordano, L., V. Gliozzi and G. L. Pozzato, *KLMLean 2.0: A Theorem Prover for KLM Logics of Nonmonotonic Reasoning*, in: N. Olivetti, editor, *Proceedings of TABLEAUX 2007 (16th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods)*, LNAI **4548** (2007), pp. 238–244.

[6] Hustadt, U., D. Tishkovsky, F. Wolter and M. Zakharyaschev, *Automated reasoning about metric and topology.*, in: *JELIA'06*, LNAI **4160** (2006), pp. 490–493.

[7] Kraus, S., D. Lehmann and M. Magidor, *Nonmonotonic reasoning, preferential models and cumulative logics*, Artificial Intelligence **44** (1990), pp. 167–207.

[8] Kurucz, A., F. Wolter and M. Zakharyaschev, *Modal logics for metric spaces: Open problems.*, in: S. N. Artëmov, H. Barringer, A. S. d'Avila Garcez, L. C. Lamb and J. Woods, editors, *We Will Show Them! (2)* (2005), pp. 193–108.

[9] Lehmann, D. and M. Magidor, *What does a conditional knowledge base entail?*, Artificial Intelligence **55** (1992), pp. 1–60.

[10] Lewis, D., "Counterfactuals," Basil Blackwell Ltd, 1973.

[11] Nute, D., "Topics in Conditional Logic," Reidel Publishing Company, Dordrecht, 1980.

[12] Olivetti, N. and G. L. Pozzato, *CondLean: A Theorem Prover for Conditional Logics*, in: M. Cialdea Meyer and F. Pirri, editors, *Proceedings of TABLEAUX 2003 (Automated Reasoning with Analytic Tableaux and Related Methods)*, LNAI **2796** (2003), pp. 264–270.

[13] Olivetti, N. and G. L. Pozzato, *CondLean 3.0: Improving Condlean for Stronger Conditional Logics*, in: B. Beckert, editor, *Proceedings of TABLEAUX 2005 (Automated Reasoning with Analytic Tableaux and Related Methods)*, LNAI **3702** (2005), pp. 328–332.

[14] Olivetti, N. and G. L. Pozzato, *Theorem Proving for Conditional Logics: CondLean and GoalDuck*, Journal of Applied Non-Classical Logics (JANCL) **18** (2008), pp. 427–473.

[15] Sheremet, M., D. Tishkovsky, F. Wolter and M. Zakharyaschev, *Comparative similarity, tree automata, and diophantine equations*, in: G. Sutcliffe and A. Voronkov, editors, *LPAR 2005*, Lecture Notes in Computer Science **3835** (2005), pp. 651–665.

[16] Sheremet, M., D. Tishkovsky, F. Wolter and M. Zakharyaschev, *A logic for concepts and similarity*, J. Log. Comput. **17** (2007), pp. 415–452.

[17] Sheremet, M., F. Wolter and M. Zakharyaschev, *A modal logic framework for reasoning about comparative distances and topology* (2009), to appear in Annals of Pure and Applied Logic, 2009.

[18] Stalnaker, R., *A theory of conditionals*, In N. Rescher (ed.), Studies in Logical Theory, American Philosophical Quarterly, Monograph Series no.2, Blackwell, Oxford (1968), pp. 98–112.